UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/998,511	11/30/2001	Jeremy Alan Arnold	IBM / 194	6560

26517 7590 01/13/2006

WOOD, HERRON & EVANS, L.L.P. (IBM)
2700 CAREW TOWER
441 VINE STREET
CINCINNATI, OH 45202

EXAMINER	
PHAM, CHRYSTINE	

ART UNIT	PAPER NUMBER
2192	

DATE MAILED: 01/13/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MAILED

JAN 1 3 2006

Technology Center 2100

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 09/998,511
Filing Date: November 30, 2001
Appellant(s): ARNOLD ET AL.

Scott A. Stinebruner
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed October 3, 2005 appealing from the Office action mailed May 4, 2005.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

U.S. Patent No. 5,845,125	NISHIMURA ET AL.	Dec. 1, 1998
U.S. Patent No. 5,093,914	COPLIEN ET AL.	Mar. 3, 1992
U.S. Patent No. 5,740,440	WEST	Apr. 14, 1998

Appellants' Admission of Prior Art (Nov. 30, 2001), Background of the invention, page 2.

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claims 1, 3-4, 7-10, 13, 15-18, 20, 22-24, 26-30 stand rejected under 35 U.S.C. 102(b) as being anticipated by U.S. Patent No. 5,845,125 to Nishimura et al. (hereinafter, Nishimura).

Claims 2, 5, 14, 19, 21 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Nishimura in view of U.S. Patent No. 5,093,914 to Coplien et al.

Claim 11 stands rejected under 35 U.S.C. 103(a) as being unpatentable over Nishimura in view of U.S. Patent No. 5,740,440 to West.

Claims 12 and 25 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Nishimura in view of Appellants' Admission of Prior Art.

This rejection is set forth in the final Office Action mailed on May 4, 2005.

(10) Response to Argument

Essentially, Appellants' arguments, which are presented in the Appeal Brief, are the same arguments presented in the amendments filed November 29^{th} 2004, i.e., before the final Office Action.

(A) <u>Claim 1</u>

The Appellants generally argue that Nishimura of record does not teach the limitations of claim 1, i.e., "(a) in response to user input, setting an inheritance breakpoint that is associated with a first program entity in the object-oriented computer program and a method identified in the first program entity; and (b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from and that depends from the first program entity" (Brief, starting on page 5, last paragraph).

(a) More specifically, The Appellants conclude that "Nishimura is specifically directed to setting breakpoints on individual objects, or instances, of a class, **rather than upon all instances of a class**" and that "the use of object-type breakpoints teaches the opposite scenario to that recited in claim 1" (Emphasis added)(Brief, page 6, 2^{nd} and 3^{rd} full paragraphs). The Appellants further add, "Specifically, claim 1 recites

halting execution at an implementation of a method that is defined in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated" (Brief, page 6, 3rd full paragraph).

- First, the Examiner strongly and respectfully disagrees with Appellants' conclusion, based on such characterization of Nishimura. Second, it should be noted that Appellants' conclusion clearly contradicts with Appellants' own understanding of Nishimura, as illustrated in the example given by the Appellants on page 8, 2nd paragraph. On page 8, 2nd paragraph, Appellants specifically characterize Nishimura's teaching, "As an example, if there is a **base class A**, two child classes B and C that inherit from class A, a grandchild class D that inherits from class B, and a grandchild class E that inherits from class C, if a **breakpoint is set on an operation in class A**, the **breakpoint will be hit any time that operation is called for any objects created from any of classes A-E**" (Emphasis added). Needless to say, **class A** anticipates the "first program entity", each of the **classes B-E** anticipates "second program entity that depends from the first program entity", **a breakpoint is set on an operation in class A** anticipates "first program entity with which an inheritance breakpoint is associated", and **breakpoint will be hit any time that operation is called for any objects created from any of classes A-E** anticipates "halting execution at an implementation of a method that is defined in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated". Thus, contrary to Appellants conclusion that Nishimura teaches the

"opposite scenario to that recited in claim 1", Nishimura clearly teaches precisely the same scenario recited in claim 1, as pointed out by Appellants in the given example. For further discussion, see section **(d)** below.

- Once again, as has been established in the final Office Action (pages 2 and 4), the teaching of Nishimura is not limited to using the "object-type" breakpoints as alleged. On the contrary, in col.2:8-4:40, Nishimura discloses setting inheritance breakpoint in a method of the **base class** ***class [array]*** (i.e., **first program entity**) (see at least *base class, class [array], class [array operation* col.2:8-col.4:40), and halting execution of the program in response to reaching an implementation of the method (see at least *execution, object "a", class [array] operation*, *indirect base classes, descendant class* col.4:9-40) defined in the class ***class [character-string]*** (i.e., **second program entity that is different from the first program entity**) deriving from (i.e., **depends from**) the **base class *class [array]***(see at least *derived class*, *class [character-string]* col.2:8-col.4:40). Furthermore, col.14:55-col.15:5 of Nishimura specifically reads:

The user can set a breakpoint on an object basis. That is, when the user specifies an object, where a breakpoint is to be set, via the user interface section 6, the breakpoint setting section 16 sets a breakpoint in that specified object. As in a conventional debug system, the user **can also** specify only an **address** to **set a breakpoint** that is ***applied to all objects***. The user can set a breakpoint at a desired location in an object through the breakpoint setting section 16. That is in the debugger in embodiment 1, the user can set **one** of **two types of breakpoint**: object-type breakpoint and (object-specified) **address-type breakpoint**. An "object-type breakpoint" is used to break the execution of a user-specified

> object when an operation is performed on the object, while an "**address-type breakpoint**" is used to **break the execution** of an object **when control reaches a user-specified location** in a user-specified object (that is, at a **location in a particular member function or in a particular location in an inherited member function**)(Emphasis added).

It is very clear, from the above passage, that the address-type breakpoint is set in an inherited member function of a class (i.e., program entity) so that *execution* of **<u>all objects</u>**, i.e., instances, of the class is *halted* when program control reaches the user-specified location (i.e., the address-type breakpoint set in the inherited member function of the class). It is also clear that the inherited member function is an implementation of the function (i.e., method), which has already been identified (i.e., associated with) in the base class (i.e., first program entity) and further defined in the class. Thus, Nishimura setting a user-specified address-type breakpoint in an **inherited** member function clearly anticipates "in response to user input, setting an inheritance breakpoint that is associated with a first program entity in the object-oriented computer program and a method identified in the first program entity". Furthermore, in col.15:22-26, Nishimura specifically states:

> When **the class of the object** <u>inherits</u> other multiple-level classes via the **base class**, **breakpoints are set**, using this inheritance relation, in **all** the public member functions in **all** the **classes** which are inherited.(Emphasis added)

Let the class that inherits other classes be **subclass A** (i.e., **second program entity**). Again, it should be clear, from the above passage, that the **base class** is the **first program entity** from which **subclass A** (i.e., **second program**

entity) depends. And setting breakpoints in **all** functions of the **base class** (i.e., **first program entity**) (see at least *User Breakpoint Setting, address-type breakpoint, **applied to all objects**, execution, control, user-specified location, inherited member function* col.14:55-col.15:25) clearly anticipates setting breakpoints on all functions *to be inherited in subclass A*. Thus, when control reaches an inherited function in subclass A, execution is halted because a breakpoint has been set in the implementation of the same function from the base class. Thus, Nishimura clearly anticipates "halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from and that depends from the first program entity". In view of the foregoing discussion, not only does Nishimura teach setting breakpoints on an object basis, Nishimura discloses setting inheritance breakpoints on methods of the subclass (i.e., second program entity) and its base class(es) (i.e. first program entity) so that execution is halted when the methods are invoked on **all objects** of the class.

(b) The Appellants attempt to argue that Nishimura does not teach halting the program in response to reaching an implementation of a method in the subclass (i.e., "second program entity") that **overrides** the method identified in the base class (i.e., "first program entity"). Specifically, the Appellants contend, "consider a program including three classes, A, B, and C, where class B is a

subclass of class A, and **class C** is a **subclass of class B**. Consider also that **class B** has a **method X**, and **class C overrides this method** with a **method X'**. Using the invention recited in claim 1, a user may be permitted to set an inheritance breakpoint on method X of class B, and have the program halted upon encountering method X' in class C. Conversely, in the Nishimura system, a user may be permitted to set a breakpoint on an instance of class B, and have a breakpoint automatically set in class A. Given that class C depends from class B, however **Nishimura would not automatically set any breakpoint on an instance of class C as a result of setting the breakpoint on the instance of class B**" (Brief, page 6, last paragraph through page 7, 2nd full paragraph).

➢ Once again, the Examiner strongly and respectfully disagrees with Appellants' characterization of Nishimura. First, as discussed above in **(a)**, Nishimura clearly teaches setting a breakpoint on a method of a base class (as supposed to an instance of a class) and halting execution in response reaching an implementation of the method defined in the subclass. Second, it should be noted that the concept of "**overriding**" a method of the base class, e.g., X, in the subclass, e.g., with a method X', is not clearly reflected in the limitations recited in claim 1. Nor is *automatically setting breakpoint on the 'overriding' method (in the second program entity) as a result of setting the inheritance breakpoint associated with the first program entity* reflected in the limitations, as claimed. Claim 1 merely recites:

(a) in response to user input, **setting an inheritance breakpoint** that is associated with a **first program entity** in the object-oriented computer program

and a **method identified in the first program entity**; and (b) **halting execution** of the object-oriented computer program during debugging **in response** to **reaching an implementation of the method defined in a second program entity** in the object-oriented computer program that is different from and that depends from the first program entity" (Emphasis added)

Thus, contrary to Appellants' assertion, claim 1 does not recite **automatically set any breakpoint on an implementation of the method defined a second program entity** as a result **of setting the breakpoint on the method identified in the first program entity**, or *something to that effect*. If anything, claim 1 merely calls for halting execution upon **reaching the implementation of the method in the second program entity** as a result of **setting an inheritance breakpoint** that is associated with the method, identified in the **first program entity**. Furthermore, merely claimed as "an implementation of the method **defined** in a second program entity" (Emphasis added), the implementation of the method is not limited to an implementation that **overrides** the method, because "an implementation of the method defined in a second program entity" as claimed, reads on an inherited function (i.e., a function **defined** in the second program entity **as inherited** from the first program entity) of Nishimura since an inherited function is inherently **defined** in the second program entity to be the same as that **defined/identified** in the base class (i.e., "first program entity"). This also explains why it is possible in Nishimura that when a breakpoint is set in the method of base class A, execution is halted when the method is invoked on any objects of any subclasses of class A, as Appellants have pointed out in the Appellants' own example. See above **(a)** or below **(d)**.

(c) It should be noted that, in the argument against the rejection of claim 1, the Appellants appear to be confused about Nishimura's disclosure. For example, the Appellants, first referring to col.15:11-13 and FIG.9 of Nishimura, conclude that "the use of object-type breakpoints results in the **automated** setting of breakpoints in all of the public functions of a class and of any base (parent) classes therefor" (Emphasis added)(Brief, page 6, 3rd full paragraph), then referring to col.4:9-20 (Nishimura's discussion of **prior art**), the Appellants conclude that "the operations disclosed are **manual** operations where individual breakpoints are set, which falls far short of disclosing the features recited in claim 1" (Emphasis added)(Brief, page 7, last paragraph).

- However, as noted above, in col.4:9-20, Nishimura is only discussing the **prior art**, in which the user must set a breakpoint not only in the class [character-string] operation but also in the operation inherited from the class [array] in order to break execution on an object of type [character-string]. In other words, Nishimura is only identifying the problem of the prior art, for which his invention can overcome, that is to say, **automatically** setting breakpoints in all public functions of a class and all base classes, instead of requiring the user to identify all the inherited functions and base classes in order to set breakpoints therein.

(d) The Appellants, referring to col.4:21-33 of Nishimura, similarly assert that Nishimura is only directed to "setting breakpoints on **base classes** for a **particular object**" (Emphasis added)(Brief, page 8, first paragraph). It should be noted that Appellants use the term "base classes for a particular object" throughout the arguments to, perhaps, presumably suggest that Nishimura does not teach "halting execution on **all objects**/instances of a class".

➢ However, it is submitted that, an **object** in an object-oriented environment belongs to a **particular class** (of which the object is a type). Thus, "base classes for a particular object" is the same as ***base classes** for the **particular class** of which the **object** is a type*. It is important to note that in the argument, Appellants, referring to col.4:34-40 of Nishimura, state that "this disclosure simply acknowledges that when a <u>breakpoint is set</u> in a **base class** of a particular object, <u>when that breakpoint is hit</u>, the **object** that <u>hit the breakpoint</u> might be the **object that inherits from the base class**. As an example, if there is a **base class A**, two child classes B and C that inherit from class A, a grandchild class D that inherits from class B, and a grandchild class E that inherits from class C, if a **breakpoint is set on an operation in class A**, the **breakpoint will be hit any time that operation is called for any objects created from any of classes A-E**" (Emphasis added)(Brief, page 8, 2nd paragraph). It is submitted that, the above example, as understood and given by Appellants to rebut against Nishimura, serves to illustrate that Nishimura clearly anticipates every limitation of claim 1, that is to say, "(a) in response to user

input, setting an inheritance breakpoint that is associated with a first program entity (i.e., **base class A**) in the object-oriented computer program and a method identified in the first program entity (i.e., **breakpoint is set on an operation in class A**); and (b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity (i.e., any of **classes B-E**) in the object-oriented computer program that is different from and that depends from the first program entity". Thus, in response to Appellants' argument that, referring to col.14:55-col.15:25 of Nishimura, "this passage states explicitly that 'when the user selects an object-type breakpoint, the breakpoint setting section 16 sets a breakpoint in all the public functions of the class and of the indirect base classes' (col.15:11-13). Thus, this passage of Nishimura does not support the Examiner's position, and as Applicants have noted above, discloses precisely the opposite configuration to that recited in claim 1" (Brief, page 8, last paragraph), it is submitted that the Examiner's position is fully supported by Nishimura teaching setting breakpoints on all functions of the subclass (i.e., second program entity) setting breakpoints on all functions of the base classes so that execution is halted when the functions are called for any objects of the subclass or the base classes, as well as by the Appellants for acknowledging such teaching.

(e) Similarly, Appellants continue to argue that "Nishimura, on the other hand, is directed to halting execution of a program base upon a particular

instance of an object, rather than halting on every instance of the object" (Brief, page 9, second full paragraph). Once again, as discussed in **(d)**, Appellants' own example, which characterizes the teaching of Nishimura, clearly contradicts with the above allegation.

(B) Claim 8

The Appellants point to FIG.9, col.4:21-33, col.15:11-14, and col.15:23-26 of Nishimura stating that "in each of these passages it is quite evident that Nishimura discloses the concept of identifying those methods that are inherited from base classes and setting breakpoints on those methods – once again the precisely opposite configuration to that recited in claim 8" (Brief, page 10, first full paragraph). It should be noted that claim 8 specifically recites:

> "The computer-implemented method of claim 1, further comprising, during loading of a class in the object-oriented computer program, identifying each **implementation of the method** in the class and **setting a breakpoint on such implementation**, wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method includes reaching a breakpoint set on such implementation"(Emphasis added).

As discussed above with respect to claim 1, Nishimura expressly discloses setting breakpoints on all public functions (including inherited functions) of a class, as well as all public functions of base classes from which the class inherits (i.e., "depends"). Thus, contrary to Appellants assertion, Nishimura teaches precisely the configuration recited in claim 8. The Appellants further attempt to distinguish the claimed limitation "implementation of a method" over prior art by arguing that "An **implementation** of a

method the context of the invention is not found in a **base class**" and that "implementations" of a method in a class, is described in page 8, lines 18-21 of the Application as being methods defined in classes that implement abstract classes (Brief, page 10, first full paragraph). It should be noted that the features upon which Appellants rely (i.e., implementations of a method are not found in a base class **and** only found in the subclasses) are not recited in the rejected claim(s). Furthermore, merely recited as "implementation of the method in the class", it is not clear from the plain language of claim 8 that "the class" has to be a subclass (i.e., "second program entity" recited in claim 1) and not just any class (i.e., including base classes or "first program entity" recited in claim 1). Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993). Furthermore, Appellants' assertion that "An **implementation** of a method the context of the invention is not found in a **base class**" fully contradicts with the limitations of claim 3, which specifically recites:

> "The computer-implemented method of claim 1, wherein the **first program entity** is a **first class** that includes a second **implementation of the method**, wherein the **second program entity** is a **second class** that inherits from the **first class**, and wherein the first implementation of the method in the second class overrides the second implementation of the method in the first class"(Emphasis added).

Needless to say, the Specification does not limit the "implementations of a method" to just those physically found subclasses, but also, base classes (that are not necessarily abstract or interface) from which the subclasses inherit.

(C) <u>Claim 9</u>

The Appellants present the same argument as has been addressed for claim 8 above, i.e., the Appellants similarly argue that the cited passages (FIG.9, col.4:21-33, col.15:11-14, and col.15:23-26 of Nishimura) teaches "precisely opposite configuration to that recited in claim 9" (Brief, page 10, last paragraph). Again, as discussed above with respect to claim 8, the limitation "implementations of a method" as claimed, does not limit the "implementations of a method" to just those implementations physically found in the subclass (i.e., "second program entity"). Thus, as discussed above in connection to claim 8, Nishimura clearly anticipates the claimed limitations.

(D) <u>Claim 13</u>

The Appellants present the same argument, which as been addressed above in **(A)**. Thus the Appellants are referred to **(A)** for Examiner's response.

(E) <u>Claim 15</u>

The Appellants argue that Nishimura teaches the opposite scenario to that recited in claim 15, i.e., "setting of the breakpoint in the entity that depends from the other entity is in response to user input to set an inheritance breakpoint on the method in the other entity" (Brief, page 12, 2nd paragraph). In other words, the Appellants argue that Nishimura does not teach *setting a breakpoint in the subclass (i.e., second program*

entity that depends from the first program entity) ***in response to user input*** *to set a breakpoint in a method of the base class (i.e., first program entity).* It is submitted that, this limitation is not included in the plain language of claim 15. Claim 15 merely recites:

> "... (a) receiving user input to halt program execution during debugging in response to reaching **any** of **a plurality of implementations of a method** in an object-oriented computer program, wherein the user input to halt program execution includes user input to set an inheritance breakpoint on the method, wherein the inheritance breakpoint is **associated** with a first program entity, and wherein **at least one** of **the plurality of implementations of the method** is defined in a second program entity that depends from the first program entity; and
> (b) thereafter setting a breakpoint for **at least a subset** of **the plurality of implementations including the implementation defined in the second program entity** such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set"(Emphasis added)

As claimed, the limitation "*wherein the user input to halt program execution includes user input to set an inheritance breakpoint on the method, wherein the inheritance breakpoint is* ***associated*** *with a first program entity*" fails to limit the inheritance breakpoint to a breakpoint that is to be set **only** in a method physically found in the first program entity (i.e., base class) as argued. Merely claimed as being ***associated*** with a first program entity (i.e., base class), *the inheritance breakpoint to be set on the method* is not limited to *a breakpoint that is set on the method* ***physically found*** *in the first program entity (i.e., base class).* Thus, setting an inheritance breakpoint on a method is merely setting a breakpoint on a method that is associated with the first program entity (i.e., inherited function). Moreover, the limitations "***any*** *of* ***a plurality of implementations of a method*** *in an object-oriented computer program*" and "*wherein* ***at least one*** *of* ***the plurality of implementations of the method*** *is defined in a second*

<u>*program entity*</u> *that depends from the first program entity*" fail to limit the **plurality of implementations of the method** to include <u>only</u> those implementations found in the second program entity (i.e., subclass). In other words, the **plurality of implementations of a method**, as claimed, includes <u>**both**</u> implementations that are found in the second program entity (i.e., subclass) as well as those found in the first program entity (i.e., base class). As such, the limitation "<u>*setting a breakpoint*</u> *for* <u>***at least a subset***</u> *of* ***the plurality of implementations including the implementation defined in the second program entity***" does not limit the breakpoint setting to be performed <u>only</u> on those implementations defined (i.e., found) in the subclass since a <u>subset</u> of the **plurality of implementations of a method** may include implementations that are found in the first program entity as well as the second program entity as discussed above. Thus, since Nishimura teaches a user specifying an address to set a breakpoint (i.e., address-type breakpoint) that is applied to all objects of a class which triggers the breakpoint setting section 16 to set breakpoints on **all** public functions of a subclass and **all inherited functions** of the base classes, Nishimura clearly anticipates

> "(a) receiving user input to halt program execution during debugging in response to reaching <u>**any**</u> of **a plurality of implementations of a method** in an object-oriented computer program, wherein the user input to halt program execution includes <u>user input to set an inheritance breakpoint on the method</u>, wherein the <u>inheritance breakpoint</u> is <u>**associated**</u> with a <u>first program entity</u>, and wherein <u>**at least one**</u> of **the plurality of implementations of the method** is <u>defined in a second program entity</u> that depends from the first program entity; and
> (b) thereafter <u>setting a breakpoint</u> for <u>**at least a subset**</u> of **the plurality of implementations including the implementation defined in the second program entity** such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set"(Emphasis added)

(F) Claims 18 and 28

Since the Appellants refer to arguments, which have been addressed for claim 1 to generally argue that Nishimura does not teach the claimed limitations (Brief, page 13, first paragraph), the Appellants are referred to discussions of claim 1 set forth above in **(A)**.

(G) Claims 23-24

Since the Appellants refer to arguments, which have been addressed for claims 8-9 to generally argue that Nishimura does not teach the claimed limitations (Brief, page 13, 3rd paragraph), the Appellants are referred to discussions of claims 8-9 set forth above in **(B)-(C)**.

(H) Claims 26 and 30

Since the Appellants refer to arguments, which have been addressed for claim 15 to generally argue that Nishimura does not teach the claimed limitations (Brief, page 13, last paragraph), the Appellants are referred to discussions of claim 15 set forth above in **(E)**.

(I) Claims 2, 5, 14, 19, and 21

The Appellants generally argue that the rejection of these claims did not establish a prima facie showing of obviousness" (Brief, page 14, second paragraph). As has

been established in the final Office Action (page 3), the test for obviousness is not whether the features of a secondary reference (i.e., setting an inheritance breakpoint that is associated with an interface) may be bodily incorporated into the structure of the primary reference (i.e., setting an inheritance breakpoint that is associated with a first program entity); nor is it that the claimed invention must be expressly suggested in any one or all of the references. Rather, the test is what the combined teachings of the references would have suggested to those of ordinary skill in the art. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981). Furthermore, in object-oriented programming, an **interface** is inherently a <u>class</u> or a "**first program entity**" from which a subclass or "second program entity" depends. In addition, as has been established in the final Office Action (pages 7-8), Coplien expressly discloses *setting a breakpoint in a method of the class and halting execution during debugging in response to reaching an implementation of the method for* ***all objects*** *of the class* (e.g., col.8:45-53). Again, in col.13:62-col.14:20, Coplien expressly discloses, "it is desired that a **breakpoint** fires whenever the **f operation** is invoked on **any object** of **class A**. This <u>extends</u> to **invocations of f** on **objects** of those **subclasses of A** which themselves do not redefine f... ". It is clear from this passage that **class A** anticipates "first program entity" and the **subclasses of A** anticipate "second program entity which is different and that depends from the first program entity". It is also clear from the passage that setting a breakpoint on the f operation so that the breakpoint fires whenever the f operation is invoked on any object of class A (i.e., base class or "first program entity") as well as any object of the subclass(es) of A (i.e., "second program entity") anticipates

setting a breakpoint in a function of the first program entity and halting execution during debugging in response to reaching an implementation of the method defined in a second program entity that is different from the first program entity. As has been established in the final Office Action (page 7), FIG. 5 of Coplien further discloses said base class (i.e., "first program entity") being an class Window, i.e., **abstract class** or an **interface** (see *class Window, Template For Things Common To All Windows* FIG.5 & associated text) while said subclass(es) (i.e., "second program entity") being classes Xwindow and SunviewWindow. Needless to say, Nishimura and Coplien are analogous art since they're both directed to setting an inheritance breakpoint in a method, said breakpoint being associated with a first program entity and halting execution in response to reaching an implementation of the method in a second program entity. Thus, as has been established in the final Office Action, it would have been **obvious** to one of **ordinary skill** in the pertinent art at the time the invention was made to incorporate the teaching of Coplien into that of Nishimura for the inclusion of an abstract class or an interface. Since interfaces and abstract classes, as well known in the art, are designed for inheritance which enables data/variables and methods from the base/superclass (i.e., abstract or interface) to be automatically inherited by the derived/subclass without having to define the same data/variables and methods in the subclass, thus minimizing the amount of coding required in implementing the subclass. Thus, in the case of debugging the object-oriented classes, the motivation for the combined teaching would have been clearly to eliminate the need to manually and individually setting a breakpoint in each of the implementations of the f operation (i.e.,

method or function) found in the subclasses of A, when a breakpoint has already been set in the f operation of the class A.

(J) Claim 11

The Appellants generally contend, "claim 9 is patentable over Nishimura for the same reasons as presented above for claims 1 and 9" (Brief, page 16, second paragraph) without providing specific arguments against the disclosed teaching i.e., "setting breakpoint on a method call to an implementation of a method", for which West was relied upon. As discussed above in connection to claims 1 and 9, Nishimura clearly anticipates the claimed limitations of claims 1 and 9 and the combination of Nishimura and West would render obvious the claimed limitation "setting breakpoint on a method call to an implementation of a method". As such, the rejection of claim 11 is considered proper and maintained.

(K) Claims 12 and 25

Similarly, the Appellants generally contend, "claims 12 and 25 are patentable over Nishimura for the same reasons as presented above for claims 1 and 18" (Brief, page 16, 3rd paragraph) without providing specific arguments against the disclosed teaching, i.e., "associating a user-specified condition with the inheritance breakpoint", for which Appellants' Admission of Prior Art (APA) was relied upon. As discussed above in connection to claims 1 an 18, Nishimura clearly anticipates the claimed limitations of claims 1 and 18 and the combination of Nishimura and APA would render

obvious the claimed limitation "associating a user-specified condition with the inheritance breakpoint". As such, the rejection of claims 12 and 25 is considered proper and maintained.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

The claim rejections set forth in the final Office Action mailed on May 4, 2005 are reproduced here for completeness:

Claim Rejections - 35 USC § 102

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:
A person shall be entitled to a patent unless –
(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

2. Claims 1, 3-4, 7-10, 13, 15-18, 20, 22-24, 26-30 are rejected under 35 U.S.C. 102(b) as being anticipated by Nishimura et al. (US 5845125), hereinafter, *Nishimura et al.*.

As per claim 1, *Nishimura et al.* teach an apparatus (e.g., see FIG.1 & associated text), a computer-implemented method (e.g., col.1:9-13), a program product stored in a signal bearing medium including transmission medium and recordable medium (e.g., col.9:15-22 & 28-32) for debugging an object-oriented computer program which is resident in a memory (e.g., col.2:3-7, see *source code storage section 7* FIG.1 & associated text), the method comprising:

(a) in response to user input (e.g., see *User Breakpoint Setting, user, breakpoint section 16, address-type breakpoint* col.14:54-col.15:25), setting an

inheritance breakpoint that is associated with a first program entity in the object-oriented computer program and a method identified in the first program entity (see at least *base class, breakpoint, class [array], class [array] operation* col.2:8-col.4:40; see *address-type breakpoint, inherited member function* col. 15:1-5); and

(b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different [and that depends] from the first program entity (see at least *execution, object "a", class [array] operation, indirect base classes, descendant class* col.4:9-40; *User Breakpoint Setting, address-type breakpoint, applied to all objects, execution, control, user-specified location, inherited member function* col.14:55-col.15:25).

As per claim 3, *Nishimura et al.* teach the computer-implemented method as applied to claim 1, wherein the first program entity is a first class that includes a second implementation of the method (e.g., *class [array] operation*, col.4:9-40), wherein the second program entity is a second class/subclass that inherits from the first class (e.g., see *derived/child class, base class, class [character-string], class [array]*), and wherein the first implementation of the method in the second class overrides the second implementation of the method in the first class (e.g., *inheritance, operations, additional data, existing class* col.2:20-31).

As per claim 4, it recites limitations which have been addressed in claim 3, therefore, is rejected for the same reasons as cited in claim 3.

As per claim 7, *Nishimura et al.* teach the computer-implemented method of claim 6, wherein setting the inheritance breakpoint includes storing in a breakpoint data structure (e.g., see FIG.5 & associated text) an entry that identifies the first program entity and the method (e.g., see *object identification number & constructor* col.13:29-36, see FIG.11,12 & associated text, col.15:43-49).

As per claim 8, *Nishimura et al.* teach the computer-implemented method of claim 1, further comprising, during loading of a class in the object-oriented computer program, identifying each implementation of the method in the class and setting a breakpoint on such implementation (e.g., FIG.9 & associated text, col.4:21-33, col.15:11-14 & 23-26), wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method includes reaching a breakpoint set on such implementation (e.g., see FIG.2 & associated text, col.4:34-40, col.16:4-7).

As per claim 9, it recites limitations which have been addressed in claim 8, therefore, is rejected for the same reasons as cited in claim 8.

As per claim 10, *Nishimura et al.* teach the computer-implemented method of claim 9, wherein setting a breakpoint on each implementation of the method includes setting a breakpoint on a first statement in an implementation of the method (e.g., see FIG.30 & associated text, col.1:38-41).

As per claim 13, it recites limitations which have been addressed in claim 3, therefore, is rejected for the same reasons as cited in claim 3.

As per claim 15, *Nishimura et al.* teach a computer-implemented method (see at least) of debugging an object-oriented computer program (e.g., col.2:3-7, see *source code storage section 7* FIG.1 & associated text), the method comprising:

(a) receiving user input to halt program execution during debugging in response to reaching any of a plurality of implementations of a method in an object-oriented computer program (see at least *execution, object "a", class [array] operation, indirect base classes, descendant class* col.4:9-40; *User Breakpoint Setting, address-type breakpoint, applied to all objects, execution, control, user-specified location, inherited member function* col.14:55-col.15:25); and

(b) thereafter setting a breakpoint for at least a subset of the plurality of implementations such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set (see claim 1).

As per claims 17, 18, 20, they recite limitations which have been addressed in claims 8, 1, 3 respectively, therefore, are rejected for the same reasons as cited in claims 8, 1, 3.

As per claims 22-24, they recite limitations which have been addressed in claims 7, 9, 8 respectively, therefore, are rejected for the same reasons as cited in claims 7, 9, 8.

As per claims 26-30, they recite limitations which have been addressed in claims 1, 3, 8, and 15, therefore, are rejected for the same reasons as cited in claims 1, 3, 8, and 15.

Claim Rejections - 35 USC § 103

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole

would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 2, 5, 14, 19, 21 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Nishimura et al.* in further view of Coplien et al. (US 5093914), hereinafter, *Coplien et al.*.

As per claim 2, *Nishimura et al.* teach the computer-implemented method and apparatus as applied to claim 1 above. *Nishimura et al.* do not expressly disclose the first program entity as an interface that identifies the method, and wherein the second program entity is a class that implements the method. However, *Coplien et al.* disclose an apparatus (e.g., see FIG.1,2 & associated text) and a method for debugging an object-oriented computer program (e.g., see Abstract, see FIG.10,11 & associated text) comprising a first program entity which is an abstract class (e.g., col.6:26-35, col.8:54-56, col.8:62-col.9:17, col.9:34-45) or an interface that identifies the method (e.g., see *class Window* FIG.5 & associated text) and a second program entity which is a class that implements the method (e.g., see *class Xwindow, class SunviewWindow* FIG.5 & associated text, col.9: 34-45). *Coplien et al.* further disclose setting a breakpoint in a function of the first program entity and halting execution during debugging in response to reaching an implementation of the method defined in a second program entity that is different from the first program entity (e.g., col.8:45-53). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to incorporate the teaching of *Coplien et al.* into that of *Nishimura et al.* to replace the first program entity with an object-oriented interface or abstract class which would produce the expected result with reasonable success. And the motivation for doing so would have been that since interfaces and abstract classes are designed for inheritance which enables data/variables and methods from the base/superclass (i.e., abstract or interface) to be automatically inherited by the derived/subclass without having to define the same data/variables and methods in the subclass, thus minimizing the amount of coding required in implementing the subclass, and in the case of debugging the object-oriented classes, said inheritance feature eliminates the need to manually setting a breakpoint in all implementations (e.g., in the subclass) of the methods identified in the superclass.

As per claims 5, 14, 19, 21, they recite limitations which have been addressed in claim 2, therefore, are rejected for the same reasons as cited in claim 2.

5. Claim 11 is rejected under 35 U.S.C. 103(a) as being unpatentable over *Nishimura et al.* as applied to claim 1 above, in further view of West (US 5740440), hereinafter, *West*.

As per claim 11, *Nishimura et al.* teach the computer-implemented method of claim 9. *Nishimura et al.* do not expressly disclose setting a breakpoint on a method call to an implementation of the method. However, *West* discloses a method of debugging an object-oriented program (e.g., see Abstract) wherein breakpoints are set on method calls (e.g., col.4:1-6). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to incorporate the teaching of *West* into that of *Nishimura et al.* to enable setting breakpoint on method calls with reasonable success. And the motivation for doing so would have been that setting breakpoints on a method calls enables the state of the program to be examined at the suspension of execution, and determination can be made as to whether an object in which the method call is contained is newly created so that updates can be reported to the debugger for data tracking or other debug operations.

6. Claims 12 and 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Nishimura et al.* as applied to claims 1 and 18 above, in further view of Applicant's Admission of Prior Art (hereinafter APA)(see page 2 – Background of the invention).

As per claim 12, *Nishimura et al.* teach the computer-implemented method of claim 1. *Nishimura et al.* do not expressly disclose setting the inheritance breakpoint includes associating a user-specified condition with the inheritance breakpoint, and wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method is performed only if the user-specified condition has been met. However, APA discloses associating a user-specified condition with the breakpoint , and wherein halting execution of the object-oriented computer program during debugging in response to reaching the implementation of the method is performed only if the user-specified condition has been met (e.g., see *after the breakpoint has been hit X times* pg.2:13-18). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to incorporate the teaching of APA into that of *Nishimura et al.* to obtain conditional (i.e., user-specified condition) breakpoints with reasonable success. And the motivation for doing so would have been determining and halting execution only when the user-specified condition (i.e., breakpoint has been hit X times) for a breakpoint has been met would allow the user of the debugging method more control over program suspension. That is to say, by specifying and monitoring said condition associated with breakpoint A, the user could skip over X-1 breakpoint iteration(s) (which entail X-1 suspensions of program execution)

and jump to the breakpoint of interest (i.e., when breakpoint A has been hit X times), thus making program debugging a more effective and efficient process.

As per claim 25, it recites limitations which have been addressed in claim 12, therefore, is rejected for the same reasons as cited in claim 12.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Chrystine Pham
Examiner
Art Unit 2192

CP

TUAN DAM
SUPERVISORY PATENT EXAMINER

Conferees:
Tuan Dam, SPE 2192
Wei Zhen, SPE 2191

WEI ZHEN
SUPERVISORY PATENT EXAMINER